一种两级分布调度算法

田增平 瞿兆荣

(华东计算技术研究所 上海 201800)

摘要 针对传统分布式调度算法研究中将调度策略与机制相分离的不足，本文提出了两级分布调度算法。它能较好地结合两种调度机制(远程执行与进程迁移)，提高了系统的性能。

关键词 分布式调度算法 调度机制 调度策略

A Two-Level Distributed Scheduling Algorithm

[Abstract] The previous research on distributed scheduling algorithm failed to fully separate scheduling policies from scheduling mechanisms. A two-lever distributed scheduling algorithm is proposed, which effectively combines simple scheduling policies with two scheduling machanisms, remote execution and process migration. The simulation we did shows the performance of the system is improved.

[Key words] distributed scheduling algorithm / scheduling mechanism / scheduling policy

分布式调度算法研究近 20 年来，取得了丰硕成果，对分布式系统的特性、各种调度算法适应的系统类型等都有一定的研究。概括起来有下列几点：

(1) 具有负载平衡的系统性能要远好过没有负载平衡策略的系统性能；

(2) 动态负载平衡策略的性能要优于静态负载平衡策略的性能，但是，其开销要高些；

(3) 静态负载平衡策略对系统负载的动态变化很敏感，其仅具有次优性能；

(4) SI(发送者寻找接收者)策略的性能要比 RI(接收者寻找发送者)策略的性能差；

(5) 有效的负载平衡策略并不一定需要很多系统信息。过多的全局信息会破坏系统性能；

(6) 动态负载平衡策略中使用的信息必须易于获得，过时的系统信息会破坏系统性能；

(7) 负载平衡工作会增加网络负载。

然而，以前的调度算法研究中也存在一些不足，主要表现在，偏重于单纯从策略上提高算法性能而忽略了不同实现机制的差别。在所看到的算法中，基本上都是采用单一的实现机制，(如远程执行或进程迁移)来平衡系统负载。采用单一的实现机制固然有其简单的一面，但是许多分布式系统都提供了任务调度的两种实现机制。而能很好结合远程执行和进程迁移的特点，有效地平衡系统负载的调度算法，则不多见。

在不对任务假设的调度算法中，往往是采用平衡各节点活跃进程(执行进程或等待进程)队列长度的方法来平衡其负载。但它只有在各个进程执行时间相同的情况下，才能真正平衡各节点的负载。这是由于各节点的实际负载是 CPU 的利用率，而不是其进程队列的长度。执行时间对 CPU 利用率有重要影响，而各个进程的执行时间往往是随机的，即使同一个进程，其执行时间随运行环境的不同也不同。也就是说，进程的执行时间无法在其执行之前准确估计。但是，Bryant 等人认为在进程执行中，能对其剩余执行时间给出较准确的估计。

* 国防科工委基金资助项目

收稿日期：1994-03-03

1 两级分布调度模型

本节从基本思想和调度模型两个方面来说明该算法。

1.1 两级分布调度算法的基本思想

在此算法中，事先不对进程做任何假设。但是，在进程执行的任一时刻能够估计其执行所需的剩余时间。调度所使用的机制有任务远程执行和进程迁移。运用任务远程执行，总是试图保证各节点上的活跃进程数相等，称此为一级分布调度，简称"粗调"。在各节点上以一定周期对其总的剩余执行时间进行估计，若超过阈值 T，则按对称请求的位置策略，寻找接收者(发送者)，运用进程迁移机制将进程发出称此为第二级分布调度，简称"细调"。

在各节点上，当有某任务到来时，随机试探 P 个节点，若找到某节点(Tnode)任务数比本节点任务数至少少两个，则将新到来任务在 Tnode 上远程执行，否则，任务进入局部处理队列，等待调度执行。每过一定的时间间隔 T_m 各节点估算其总的剩余计算量，若超过(低于)阈值 T，则利用对称请求策略，寻找接收者(或发送者)Tnode，若找到，则选择合适的进程迁移到 Tnode(或接收 Tnode 迁移来的进程)。

1.2 两级分布调度模型

此模型中，各节点的调度算法结构完全相同(图 1)。

(1) 任务分派器

对进入节点的任务，由任务分派器决定其远程执行还是局部执行。若远程执行，则按信息策略选定的目标节点，调用任务远程执行机制，将此任务远程启动。在以后用户与该任务的联系及最后结果的提交，都由任务远程执行机制负责。若任务为局部执行，则任务分派器将交给本节点的进程管理机制，创建进程，并最终交给局部调度器调度执行。

(2) 负载平衡器

它由传送策略和位置策略两部分组成。传送策略每过一定的时间间隔，估计本节点总的剩余计算量，并与阈值 T 比较，决定本节点的状态。若其负载平衡，则不必做任何工作。否则，调用位置策略，依靠信息策略的信息决定要迁移或获取负载的节点，并调用进程迁移机制。将任务迁移出去。以后与迁移进程的联系，都由进程迁移机制负责。若需要请求目标节点分配负载，则由进程迁移机制发出请求，并管理与迁移到来进程的源节点的联系。

(3) 局部调度器

局部调度器负责调度在局部节点上执行的进程。在此，局部调度策略采用时间片轮转法，进入系统的进程分时共享处理器。可迁移的进程仅是用户进程，系统进程不能迁移。

(4) 信息策略

它负责收集各节点的任务信息，分别提供给任务分派器和负载平衡器。

(5) 会话机制

它负责在各节点上对应的调度模块之间建立联系。

(6) 任务远程执行机制和进程迁移机制

分别负责任务远程执行和进程迁移的实现。

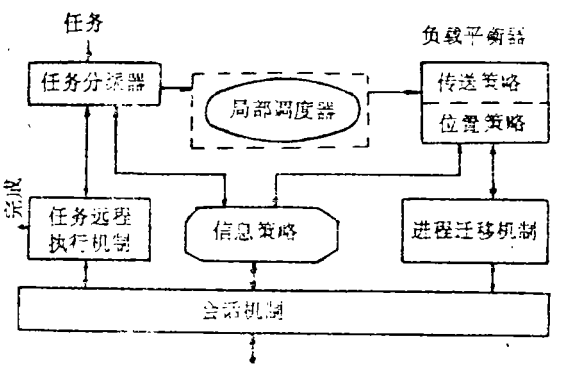

图 1 两级分布调度模型

2 调度算法的系统环境及性能目标

2.1 算法所适应的系统环境

(1) 既不考虑各处理机在处理任务时的速度差异，也忽略全局文件系统对不同节点的速度差异。

(2) 在各节点上，就绪进程按时间片轮转

法，独立调度，不考虑合作进程间的通信开销。

(3) 运行进程可迁移到其任何相邻节点，其通信费用与其代码量和通信网延迟成正比。

(4) 进程迁移不受内存限制。

(5) 各节点的传送策略和位置策略的执行开销可以忽略，但是，其处理机间的通信开销不能忽略。

(6) 每个进程的服务时间为其进入执行状态到运行结束的时间。调度算法并不知道每个进程的执行时间，但是，它可以记录每个进程在某时刻以前的执行时间，并估算其剩余执行时间(在进程开始执行之后)。

(7) 各处理机由双向点一点通信链路连接，形成一个无向图<V，E>，V 为节点集，E 为通信链路集，$<v_i, v_j> \in E$ 当且仅当 v_i 节点与 v_j 之间有直接通信链路相连。<V，E>不必是完全图，但是它必须是连通的。

(8) 任何节点在任何时刻可以接受任务。

2.2. 算法的性能目标

此算法的总目标是为了平衡各处理器的利用率:

$$\rho = \triangle R_i / \triangle t$$

△t: 为时间段;

$\triangle R_i$: 为处理器 i 在△t 中实际运行时间。

并降低任务的平均响应率:

$$RT = \sum_{n} RT_i / n$$

RT_i: 为任务 i 的响应率;

n: 为总任务数。

3 算法模拟及结果分析

讨论分布式进程调度算法，不但应从理论上说明算法性能的性能，更重要的是，要在实际应用中检验算法的性能。由于条件所限，本文讨论的算法没有具体的分布式系统可供运行，在此情况下，我们采用模拟的方法来检验其性能。

按任务来源，分布式调度算法的模拟方法可分为两大类。第一类方法称为跟踪模拟方法，其任务主要是通过对现有的分布系统进行一定时间段的跟踪，获得其在跟踪期间所实际调度的任务。再将这些任务用于要检验的调度算法来调度，观察算法的性能。这种模拟方法的最大特点是真实。因为其任务反映了实际中任务的分布情况。第二类方法可称为概率分布模拟方法，其任务是按照某种概率分布产生的。其特点是方便，无需依赖现有的分布式系统。由于条件所限，我们采用了第二种模拟方法。

3.1 系统环境

模拟采用的系统环境类似于 EtherNet 的局部网模型(图 2)。系统中共有 30 个节点(Node0～Node29)。各节点间仅通过消息传递进行通信。规定节点间的一次通信开销为 1 个时间单位，节点间进行一次任务远程分配开销为 2-3 个时间单位，节点间每进行一次迁移，开销为 3～5 倍的任务远程分配开销费用。虽然，每个系统这些参数都是不同的，对于进程迁移，其每次迁移的开销可能也是不同的，但是，由于我们要检验算法的估计特性，根据前人的工作，这样的假设是合理的。

这里采用的通信方法是点一点通信，可以为系统建立一张邻接表，以说明各节点之间的邻接关系。这里假定各节点是全连接的。

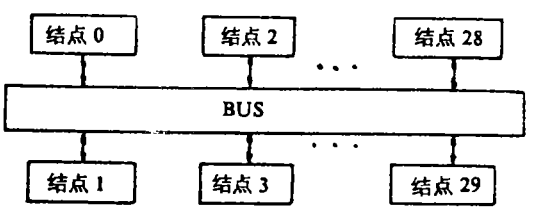

图 2 算法的模拟环境

3.2 任务环境

系统中任务的到来服从到达率为 λ 的泊松过程，即各任务到达的时间间隔分布遵循参数为 λ 的负指数分布。各任务的处理时间遵循 1～10 的随机分布。共设计了 8 组不同系统负载的任务:

T_{03}: 各节点的任务到达率相同，$\lambda_i = \lambda = 0.3$ (i = 0, …, 29)

T_{04}: 各节点的任务到达率相同，$\lambda_i = \lambda = 0.4$ (i = 0, …, 29)

T_{05}: 各节点的任务到达率相同，$\lambda_i=\lambda=0.5$ (i=0, …, 29)

T_{06}: 各节点的任务到达率相同，$\lambda_i=\lambda=0.6$ (i=0, …, 29)

T_{07}: 各节点的任务到达率相同，$\lambda_i=\lambda=0.7$ (i=0, …, 29)

T_{08}: 各节点的任务到达率相同，$\lambda_i=\lambda=0.8$ (i=0, …, 29)

T_{09}: 各节点的任务到达率相同，$\lambda_i=\lambda=0.9$ (i=0, …, 29)

T_{10}: 各节点的任务到达率相同，$\lambda_i=\lambda=3(i+1)/100$ (i=0, …, 29)

为了进行对比，我们实现了 Shivaratr 的对称请求调度算法。也记录了各节点不进行全局调度的性能。

3.3 统计参数

主要对每组任务调度的平均响应率(RT)和平均 CPU 利用率(PUR)进行了统计

平均响应率 $(RT)=\sum_i$ 各任务的响应率 (RT_i) / 任务总数

平均 CPU 利用率 $(PUR)=\sum_i$ 节点 i 的 CPU 利用率 / 节点数

4 模拟结果分析

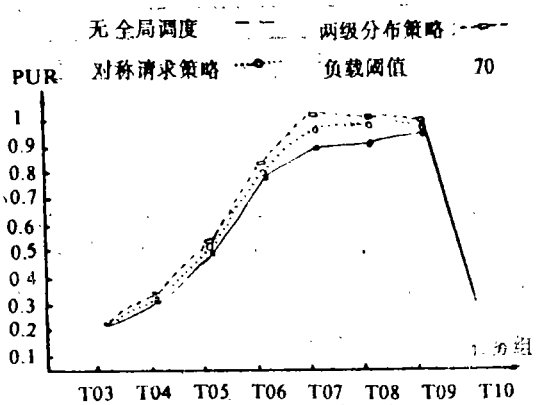

图 3 3 种策略的 CPU 利用率对比曲线

结果分析主要对 3 种算法的 CPU 利用率、平均响应率和对称请求算法与两级分布算法受负载阈值的影响进行了比较。

利用模拟结果数据可得 3 种算法的 CPU 利用率对比曲线，参见图 3

从 CPU 利用率的对比曲线上可看出：在系统负载较轻时(T03～T05)，3 种算法的 CPU 利用率都较低，而且性能差别亦不明显；当各节点负载不平衡时(T10)，各节点 CPU 利用率也相近；只有当系统负载较重时(T06～T08)，两级分布策略的性能较其它两者高，当系统负载趋于饱和时(T09)，3 种算法的 CPU 利用率又接近。3 种算法的平均响应率对比曲线如图 4 所示。

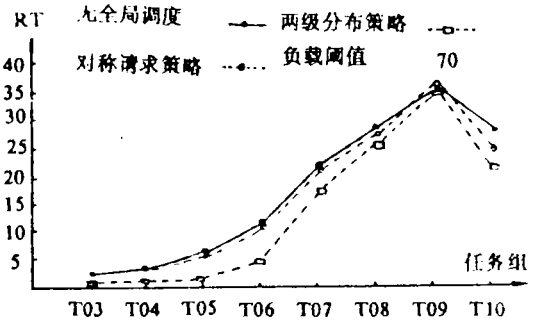

图 4 3 种策略的平均响应率对比曲线

从平均响应率对比曲线可看出：在系统轻载、中载和各节点负载不平衡时(T03～T07 和 T10)，两级分布算法的平均响应率要低于其它两者；当系统负载趋于饱和时(T08～T09)，3 种算法的平均响应率相接近。

负载阈值的影响对比曲线如图 5 所示。

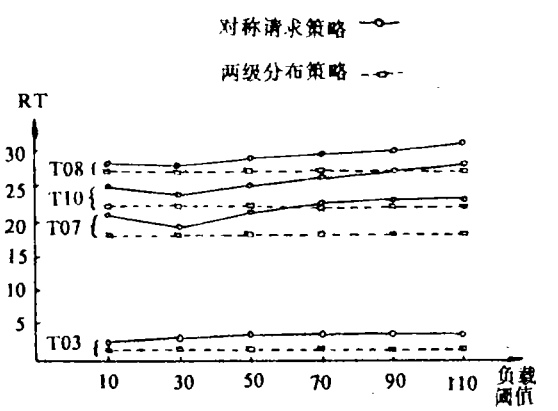

图 5 负载阈值影响的对比曲线

(下转第 10 页)

4 结束语

本文重点剖析了 Unix System V 系统调用的处理过程，从中我们可以看出：

(1) 系统调用是用户进程请求执行核心程序的唯一方式，一旦用户程序通过系统调用进入核心，便完全与用户隔离。所以 Unix 可以在系统调用层加入安全机制来对用户的存取请求做出允许或拒绝的控制，从而提高 Unix System V 的安全性。

(2) 系统调用的错误号在核心程序 systrap 中是通过一个字节转储的，即最大为 255(2^8-1)。另外，在系统头文件 errno.h 中，系统已定义了一部分错误号，所以在编写新系统调用的处理子程序中定义新的错误号时，错误号要避免与原有定义重复，且不要超过 255。

(3) systrap()是系统执行所有系统调用处理子程序的部分，所以可以在其中加入机制来对所有系统调用做一些相同的处理，比如统计某类错误，并针对某类错误做相应处理。

(4) exec()类的系统调用较特殊，系统对它做了特别处理，基于系统调用开发时，要考虑到这一点。

(5) 如果新编系统调用的一个参数为文件路径名时，要把它作为该系统调用的第一个参数。

对每一个系统调用的具体分析，这里就不再介绍了。本文的例子是在基于 Unix System V 的 386 微机上通过执行的。

参考文献

1 胡希明、张建平等. Unix结构分析. 杭州：浙江大学出版社,1990.12.

2 樊建平等. Unix System V操作系统内核代码剖析. 北京：海洋出版社,1992.5.

3 (美)莫里斯·贝奇. Unix操作系统设计. 北京：北京大学出版社,1989.11.

≈≈

(上接第 6 页)

从负载阈值的影响对比曲线来看：两级分布算法的平均响应率几乎不受负载阈值影响，在所选的各组任务中，其各种阈值的平均响应率基本相同，在图中呈现为水平线；而对称请求策略的平均响应率受负载阈值的影响较严重，在图中呈现为折线。

5 结束语

5.1 两级分布调度算法的性能

由上节的分析，可得下列结论：

(1) 采用适当的调度策略的系统的性能要优于不采用调度策略的系统性能；

(2) 两级分布调度算法比对称请求算法性能要好；

(3) 两级分布调度算法较对称请求算法受负载阈值的影响要小。

5.2 今后的工作

本文仅对两级分布调度算法进行了部分模拟试验工作，若在条件允许的情况下，应将其投入具体的系统运行，以观察其性能。对不同的系统环境，可通过大量的调度试验，确定算法的各种参数取值(如，负载阈值，负载估算周期等)。

若有可能，应建立一个统一的系统，在其中，将文中所提的算法与其它算法进行对比，并讨论其各自的适应特性。

本文的模拟工作在 AST386 微机上用 C 语言实现。

参考文献

1 Baumgartner KM. GAMMON: A Load Balancing Strategy for Local Computer Systems with Multiaccess Networks.IEEE Trans. Comput. C-38(8), 1989

2 Bryant RM, et al. A Stable Distributed Scheduling Algorithm 1981. Proc. 1th Inter. Conf. Distributed Computing System

3 Shivaratri NG, et al. Two Adaptive Location Policies for Stable Scheduling Algorithms. Proc. 10th Inter. Conf. Distributed Computing System.1990